1.) Le langage "Logo".

« Un singe et une pierre sont attachés chacun à un bout d'une corde passant sur une poulie. Le singe et la pierre sont exactement du même poids et s'équilibrent donc. Mais voilà que le singe entreprend de grimper à la corde. Qu'advient-il de la pierre ? »
Lewis Carroll.

Logo n'est pas un langage de programmation comme les autres. Capable d'enthousiasmer les plus jeunes sur les bancs de l'école, il fascine tout autant les adultes. Car, au-delà de la tortue graphique qui exécute sur l'écran d'étonnantes figures, se cache un vrai langage de programmation, riche, structuré et fonctionnel.

Les créateurs de Logo, Papert et Minski, ont souhaité créer un langage de programmation afin d'utiliser la puissance de l'outil informatique dans les tâches d'enseignement. Plus que tout autre langage, Logo a été conçu dans le but de démystifier les ordinateurs et la programmation. Tout adepte de Logo s'oppose naturellement à l'utilisation injustifiée des jargons et à toute tendance visant à faire de l'informatique un domaine à part. De ce fait, la simplicité de mise en œuvre de ce langage est étonnante : Quelques minutes suffisent pour en comprendre la logique et se lancer dans la création de projets personnels.

Pourtant, malgré cette simplicité apparente, Logo est un véritable langage de programmation qui permet d'aborder des concepts fondamentaux de l'informatique tels que la récursivité, le traitement de listes ou la pleine fonctionnalité. En cela, il se rapproche de langages utilisés par les étudiants en informatique tels que Lisp ou Caml.

Le nom « Logo » vient du grec « *Logos* » qui signifie « *parole* », « *discours* ». Les premières versions de Logo permettaient essentiellement de manipuler des mots et des phrases, mais on se rendit rapidement compte que la manipulation de symboles ne suscitait pas un grand engouement.

C'est Seymour Papert qui imagina de se servir de ce langage pour tracer des graphiques.

Logo est un langage issu le Lisp. Comme lui, c'est un langage fonctionnel. Comme Lisp, c'est un langage interprété, ce qui permet une utilisation directe sans passer par une phase de compilation.

Le premier système Logo a fonctionné en 1970 au Massachusetts Institute of Technology (MIT), dans le laboratoire d'intelligence artificielle. L'ordinateur était un PDP 10 de la société Digital Equipment Corporation. Les périphériques se composaient de deux écrans, l'un réservé au texte, l'autre au graphisme. Par ailleurs, cet ordinateur commandait les déplacements d'un petit robot, la tortue de sol (en fait un petit chariot dont les mouvements étaient commandés par des instructions introduites sur le clavier d'un ordinateur), capable de laisser une trace de ses trajets grâce à un crayon dont le maniement était prévu dans le langage.

Le Logo s'adapte à une vaste gamme d'applications qui vont de la recherche sur l'intelligence artificielle à la conception d'applications graphiques et à l'enseignement au niveau préscolaire. On a même utilisé Logo dans l'enseignement donné aux handicapés mentaux, leur permettant de contrôler cet appareil puissant qu'est l'ordinateur.

2.) Ma version du langage.

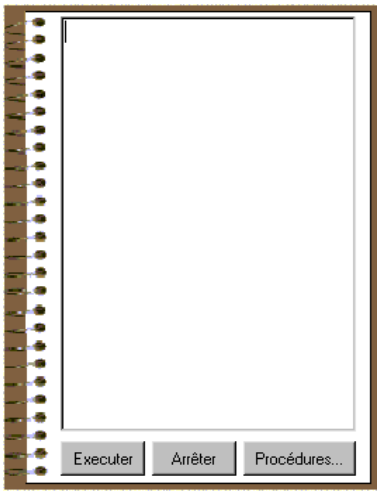
Cette version du langage Logo a été écrite en Java (version 1.1). Elle est prévue pour pouvoir être exécutée dans un navigateur Internet (les tests ont été réalisés avec MS Internet Explorer, version 4).

C'est avant tout une version ludique. Les problèmes d'optimisation, de rapidité, ont étés délaissés au profit de l'attractivité de la présentation et de la facilité d'utilisation. La tortue est aussi réaliste que possible (et que mes talents de dessinateur me le permettent).

Mais c'est aussi une version aussi complète que possible du langage Logo.

2.1.) L'interface de commande.

L'interface de commande représente (vaguement) un cahier d'écolier sur lequel on note (à l'aide du clavier) les ordres que la tortue doit exécuter.



Comme son nom l'indique, le bouton "Exécuter" lance l'exécution des commandes.

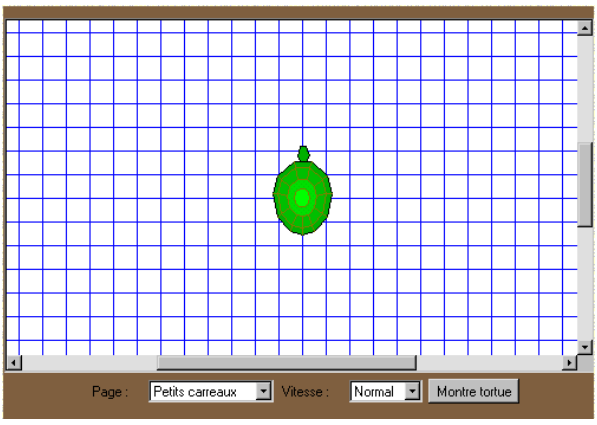
Le bouton "Arrêter", stoppe le programme en cours d'exécution (cela correspond au "CNT-C" des TO7).

Le bouton "Procédures" ouvre une fenêtre permettant la gestion des procédures définies par l'utilisateur.

> L'interface de commande est constituée par l'applet `Clogo.class`. Cette applet prend en charge l'essentiel du langage, c'est à dire l'analyse lexicale, syntaxique, l'évaluation des expressions...

2.2.) La machine "LOGO".

La machine logo symbolise une petite tortue évoluant sur une feuille de cahier d'écolier. Il est possible de modifier l'aspect de la feuille ainsi que la vitesse d'évolution de la tortue.



Le bouton "Montre tortue" permet de repositionner automatiquement les ascenseurs afin que la tortue se retrouve au centre de l'écran.

La machine Logo est constituée par l'applet logo.class. Cette applet prend en charge les aspects graphiques de Logo, c'est à dire essentiellement la représentation de la tortue de sol. Ces deux applets ("Logo.class" et "Clogo.class") doivent impérativement se trouver sur la même page HTML pour que l'ensemble puisse fonctionner.

2.3.) La gestion des erreurs.

Quand une erreur survient (que ce soit durant l'analyse des instructions ou à l'exécution), un message d'erreur est affiché à la place de la tortue. Quand cela est possible, l'instruction en cause est mise en surbrillance dans la fenêtre d'édition. Dans l'exemple ci-dessous, on a tapé « **AVONCE** *100*» au lieu de « **AVANCE** *100*».

2.4.) La syntaxe de "LOGO".

La syntaxe est une synthèse des différents Logos existants. Je me suis inspiré des Logo fonctionnant sur TO7, Apple II, PC IBM et Micral ainsi de quelques versions plus exotiques (écrites, elles aussi, en Java). Je me suis aussi inspiré de l'excellent Superlogo[1] de Longman Logotron et du MSWLogo[2] de Microsoft. Il y a aussi des apports venant d'autres langages de programmation, comme les commentaires, les calculs en notation infixée ou les nombres en notation hexadécimale.

Pas de différence majuscules / minuscules.

Afin de faciliter votre « apprentissage » du langage LOGO, et comme un bon exemple vaut mieux qu'un long discours, vous trouverez dans ce manuel de nombreux exemples de programmes. Ces programmes ont été choisis afin de bien illustrer certains aspects du LOGO

[1] Disponible en version d'évaluation (et en anglais) sur http://www.logo.com.
[2] Sur http://www.softronix.com.

3.) Les éléments du langage.

3.1.) Les commentaires.

Un commentaire commence par le symbole « // » ou par le symbole « ; ; ». Logo ignore les caractères placés après le symbole de commentaire jusqu'à la fin de la ligne.

Exemple d'utilisation de commentaires :

```
// Une procédure qui trace un carré
POUR carre :cote ;; la dimension du carré
        REPETE 4 [ AV :cote TD 90] // on trace 4 lignes perpendiculaires
FIN ;; fin de la procédure carré
```

4.) Géométrie de la tortue.

« Que nul n'entre ici s'il n'est géomètre.»
Platon.

4.1.) Déplacements.

La tortue peut être considérée comme un petit robot qui se déplace sur l'écran et que l'utilisateur peut contrôler à l'aide d'un ensemble d'ordres. A l'écran, elle est symbolisée par ce petit dessin qui indique à la fois la position et la direction de l'animal :

Cette tortue est conçue pour recevoir des ordres exprimés en « *Langage Tortue* ». On peut, par exemple lui demander d'avancer de 100 pas avec la commande "**AVANCE** *100* ", ou lui demander de se tourner à droite avec la commande "**DROITE** *90* ". Voici, par exemple, un programme élémentaire :

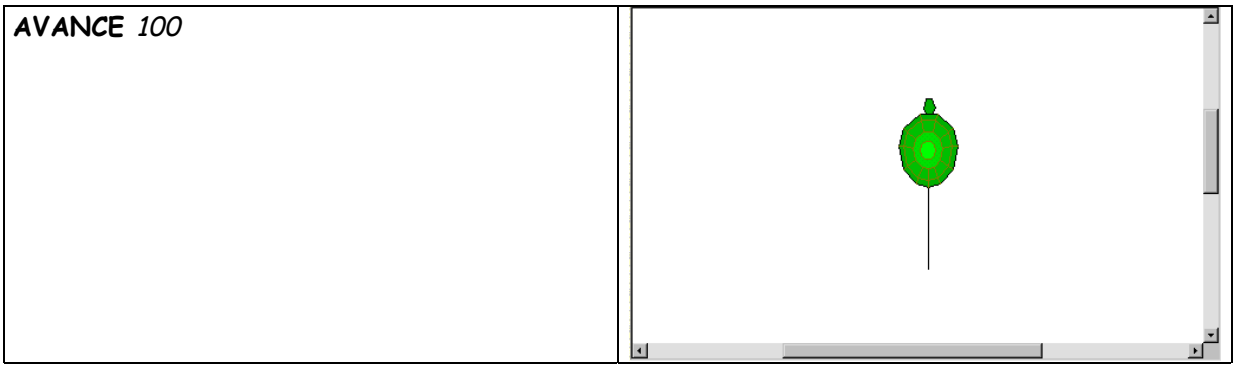
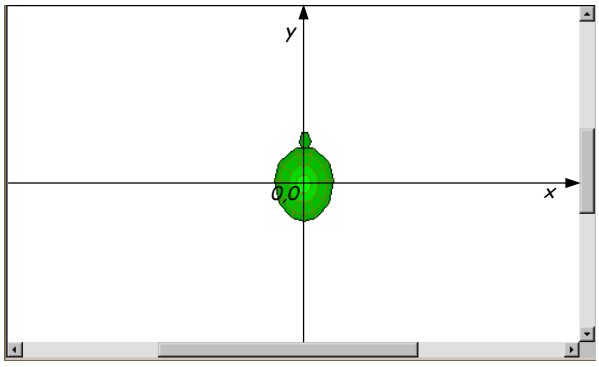


Figure 1 : Programme élémentaire.

La tortue peut laisser une trace de ses déplacements grâce à un crayon dont le maniement est prévu dans le langage. Elle peut ainsi lever le crayon (ordre **LC** ou **LEVECRAYON**), baisser le crayon (ordre **BC** ou **BAISSECRAYON**), changer la couleur du crayon (ordre **FCC** *n* ou **COULEUR** *n*).

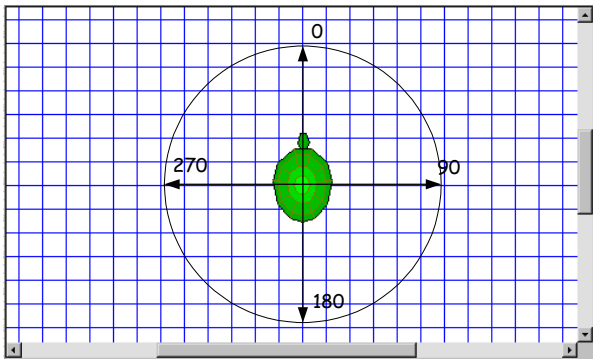
La tortue évolue dans un micromonde, un rectangle de dimension 1000*1000, constitué donc d'un million de points.



Au départ, la tortue se trouve au milieu de l'écran et se dirige vers le haut. Les coordonnées du centre sont (0,0), l'axe des *x* étant horizontal et celui des *y* vertical.

La partie visible n'est que d'environ 500*300 points mais les ascenseurs permettent de visualiser l'ensemble du champ d'activité de la tortue.

Pour les rotations, il faut indiquer un angle en degrés. Un tour complet équivaut à 360 degrés. Un demi-tour égale 180 degrés et un quart de tour égale 90 degrés.



Notez que, contrairement à la notation mathématique traditionnelle, le zéro se trouve en haut et que les angles augmentent dans le sens contraire des aiguilles d'une montre.

Bien entendu, les commandes peuvent être tapées les unes à la suite des autres. Par exemple, la suite d'ordres suivante commande à la tortue de dessiner une maisonnette :

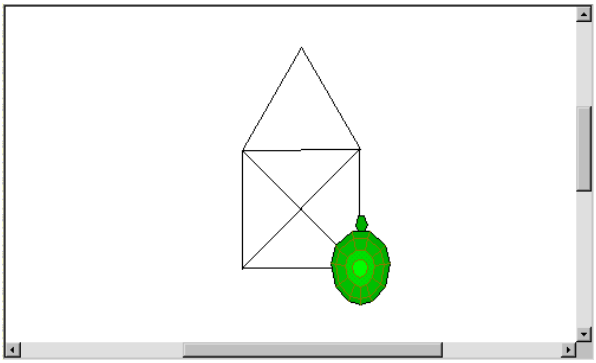
Instructions	Résultat
AVANCE *100* **DROITE** *30* **AVANCE** *100* **DROITE** *120* **AVANCE** *100* **DROITE** *120* **AVANCE** *100* **GAUCHE** *135* **AVANCE** *141.42* **GAUCHE** *135* **AVANCE** *100* **GAUCHE** *135* **AVANCE** *141.42* **GAUCHE** *135* **AVANCE** *100* **GAUCHE** *90*	

Figure 2 : Maisonnette.

Il faut toujours garder ceci à l'esprit : La tortue Logo est un animal cybernétique extrêmement introverti. Elle ne se soucie aucunement de ce qui se passe autour d'elle, elle ne fait que tracer sa ligne avec indifférence. Par exemple, si elle trace un cercle, elle ignore où se trouve le centre de ce cercle, et s'en moque éperdument. Imaginez la tortue comme un animal certes infatigable mais aussi aveugle, sourd, amnésique et un peu idiot.

4.2.) Commandes de base.

La tortue comprend également certaines abréviations : par exemple, on peut taper " **AV** *100* " au lieu de " **AVANCE** *100* " et " **TG** *135* " au lieu de " **GAUCHE** *135* ". Le tableau suivant récapitule les principaux ordres reconnus par la tortue et les variantes acceptées :

Ordre	Fonction	Synonymes
AV *n*	Commande à la tortue d'avancer de n[3] pas.	**AVANCE** *n* **VA** *n*
TD *n*	Commande à la tortue d'effectuer une rotation de n degrés à droite.	**TOURNEDROITE** *n* **TOURNE_DROITE** *n* **DROITE** *n* **DR** *n*
TG *n*	Commande à la tortue d'effectuer une rotation de n degrés à gauche	**TOURNEGAUCHE** *n* **TOURNE_GAUCHE** *n* **GAUCHE** *n* **GA** *n*
REC *n*	Commande à la tortue de reculer de n pas.	**RECULE** *n* **RE** *n*
FPOS [*n1 n2* **]**	Place la tortue à la position (n1,n2) de l'écran.	**FIXEXY** *n1 n2*
FCAP *n*	Fixe le cap de la tortue.	**FIXECAP** *n* **FIXCAP** *n* **FIXE_CAP** *n*
VE	Efface l'écran.	**VIDEECRAN** **VIDE_ECRAN** **NETTOIE**
MT	Rend la tortue visible.	**MONTRETORTUE** **MONTRE_TORTUE**
CT	Rend la tortue invisible à l'utilisateur	**CACHETORTUE** **CACHE_TORTUE**
LC	Demande à la tortue de lever son crayon.	**LEVECRAYON** **LEVE_CRAYON** **MARCHER** **LEVEPLUME** **LP**
BC	Demande à la tortue de baisser son crayon.	**BAISSECRAYON** **BAISSE_CRAYON** **DESSINER** **BAISSEPLUME** **BP**
FCC *n*	Fixe la couleur du crayon de la tortue en fonction de n : 0 ⇒ Noir 3 ⇒ Jaune 6 ⇒ Bleu clair 1 ⇒ Rouge 4 ⇒ Bleu 7 ⇒ Blanc 2 ⇒ Vert 5 ⇒ Violet Si n est supérieur à 0, alors n est équivalent à "n MOD 8". Par exemple, pour n=8, on aura la couleur "Noir", pour n=9, "Rouge", pour n=10, "Vert" etc.	**COULEUR** *n* **COULEUR_CRAYON** *n*

Tableau 1 : Commandes de base.

[3] Il est possible de rentrer les constantes numériques en hexadécimal, si elles sont précédées de "**0x**". Par exemple, "**AVANCE** *4779*" est équivalent à "**AVANCE** *0x12AB*".

4.3.) Exercices avec les commandes de base.

Exercice 1.

Dessiner un carré de coté=100 à l'aide de la tortue. Dessiner ensuite un triangle à l'intérieur de ce carré.

Exercice 2.

Dessiner un bateau (élémentaire) à l'aide de la tortue.

5.) La répétition.

La répétition est une structure de programmation qui existe dans pratiquement tous les langages évolués. Comme son nom l'indique, elle permet de répéter un certain nombre de fois des instructions. Par exemple, pour indiquer à la tortue de répéter 4 fois les instructions "**AV** *50* **TD** *90*", on note : "**REPETE** *4* **[AV** *50* **TD** *90* **]**".

Ordre	Fonction	Synonymes
REPETE *n* **[** *liste* **]**	Répète n fois les instructions contenues dans liste.	

Tableau 2 : La répétition.

Voici plusieurs exemples d'utilisation de cette structure de programmation :

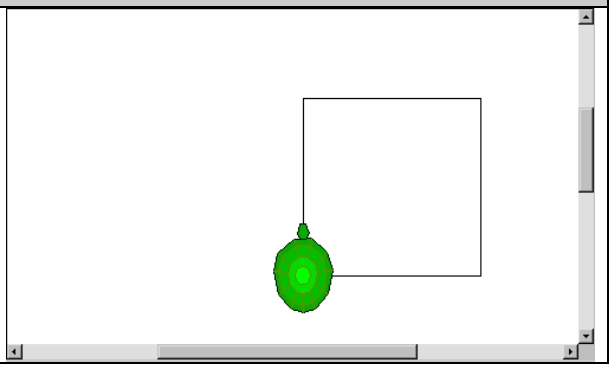
Instructions	Résultat
// Dessine un carré **REPETE** *4* **[AV** *150* **TD** *90***]**	

Figure 3 : Carré.

La liste des instructions à répéter peut contenir elle-même une ou plusieurs instructions "**REPETE**", comme dans le programme suivant :

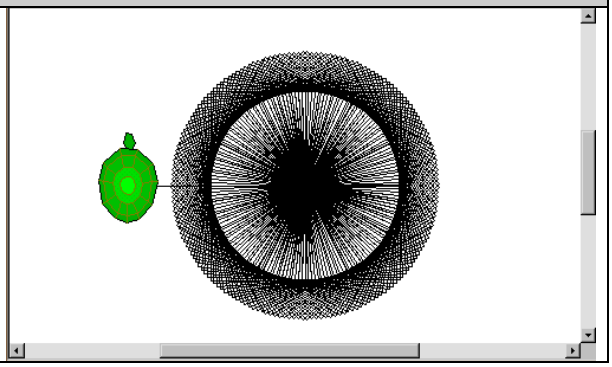
Instructions	Résultat
VE *// Efface l'écran* **REPETE** *180* **[REPETE** *4* **[AV** *80* **TD** *90* **] TD** *2* **]** **FPOS [***350 500* **] //** *Positionne la tortue*	

Figure 4 : Cercles.

Une question qui revient souvent est « Comment fait-on tourner en rond la tortue ? ». La solution est simple : pour tourner en rond, on fait un petit pas en avant, on se tourne un peu, on refait un pas en avant, on tourne encore un peu etc. De cette description au programme, il n'y a qu'un pas :

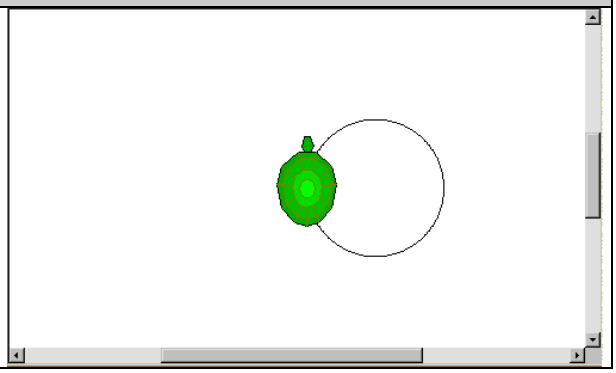
Instructions	Résultat
VE // *Efface l'écran* REPETE *360* [AV *1* TD 1]	

Figure 5 : Tourner en rond.

Ordre	Fonction	Synonymes
HASARD *n*	Renvoie un nombre compris entre 0 et n.	
CAP	Renvoie le cap de la tortue.	

Tableau 3 : Fonctions "HASARD" et "CAP".

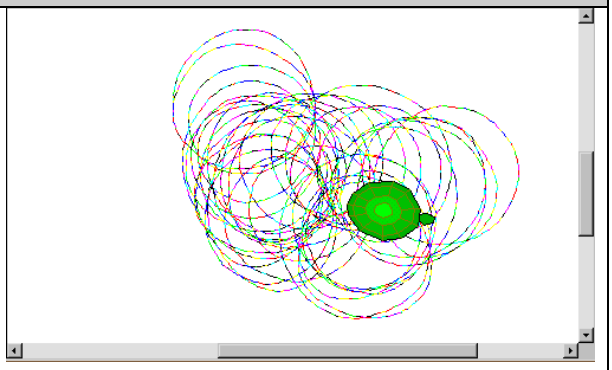
Instructions	Résultat
// *Gribouillis* VE // *Efface l'écran* REPETE *4000* [FCC HASARD *7* AV *1* + HASARD *5* TD *1* + HASARD *5*]	

Figure 6 : Gribouillis.

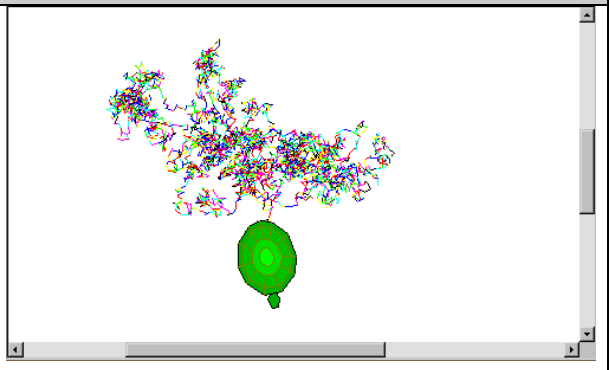
Instructions	Résultat
// *Gribouillis* VE // *Efface l'écran* REPETE *4000* [FCC HASARD *7* AV *1* + HASARD *6* FCAP HASARD *360*]	

Figure 7 : Dessin aléatoire

Quant on se trouve à l'intérieur d'une instruction **REPETE** on a accès à une variable appelée **LOOP**[4] qui renvoie le nombre de fois que la boucle a été exécutée. Cette valeur augmentera donc avec chaque répétition : au premier tour 1, puis 2, puis 3, ainsi de suite...

Ordre	Fonction	Synonymes
LOOP	Renvoie le nombre de fois que l'instruction **REPETE** a été exécutée.	

Tableau 4 : Variable LOOP.

Voici quelques exemples d'utilisation de la variable "**LOOP** " :

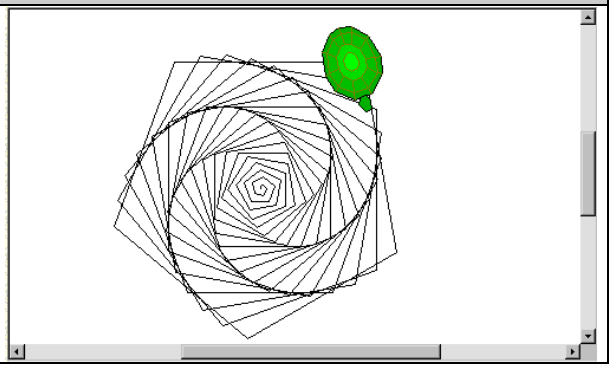
Instructions	Résultat
// Utilisation de "loop" **VE** *// Efface l'écran* **REPETE** *100* **[AV** *1.5* *** LOOP TD** *70* **]**	

Figure 8 : Utilisation de "LOOP".

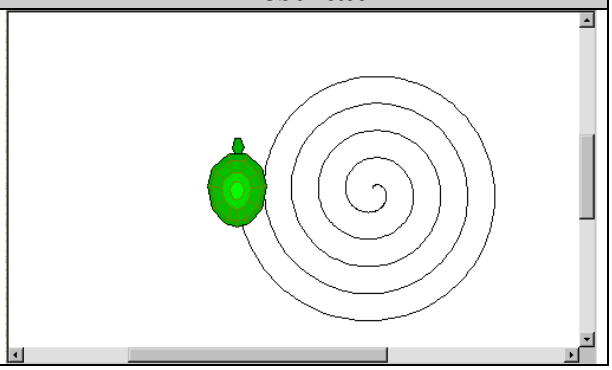
Instructions	Résultat
// Spirale **VE** *// Efface l'écran* **REPETE** *540* **[AV** *0.01* *** LOOP TD** *3* **]**	

Figure 9 : Spirale.

[4] La variable " **LOOP** " est une facilité de programmation qui n'existe pas dans le Logo original. On peut facilement la remplacer par un compteur. On pourrait ainsi modifier le programme "Spirale" de cette manière :

Version utilisant LOOP	Version n'utilisant pas LOOP
REPETE *540* **[AV** *0.01* *** LOOP TD** *3* **]**	**DONNE "CPT 1** **REPETE** *540* **[AV** *0.01* *** :CPT TD** *3* **DONNE "CPT :CPT +** *1* **]**

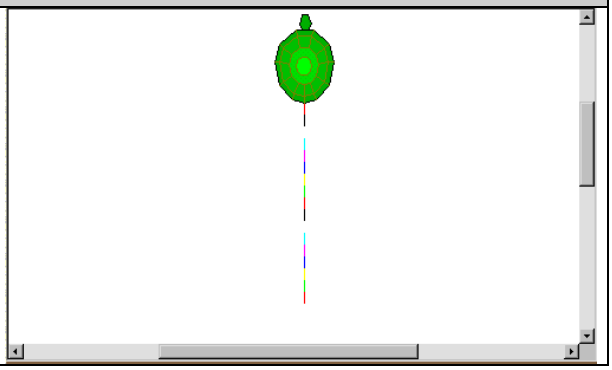
Instructions	Résultat
// Couleurs VE // Efface l'écran REPETE 20 [FCC LOOP AV 10]	

Figure 10 : Couleurs du crayon.

Bien entendu, quand plusieurs instructions **REPETE** sont imbriquées, on accède toujours à la variable **LOOP** de la boucle dans laquelle on se trouve.

5.1.) Exercices sur la répétition.

Exercice 1.

Dessiner un hexagone (une figure géométrique ayant 6 cotés) en utilisant l'ordre **REPETE**.

Exercice 2.

En s'inspirant de la spirale, dessiner un escargot.

6.) L'alternative.

Tout comme la répétition, l'alternative ou "*conditionnelle*" est une structure de programmation courante. Elle permet de conditionner l'évolution du programme.

Ordre	Fonction	Synonymes
SI *prédicat* **[***liste1***] [***liste2* **]**	Si prédicat est vrai, exécute les instructions contenues dans liste1 sinon celles de liste2. La liste **[***liste2* **]** est optionnelle.	

Tableau 5 : L'alternative.

Voici quelques exemples d'utilisation de l'alternative :

Instructions	Résultat

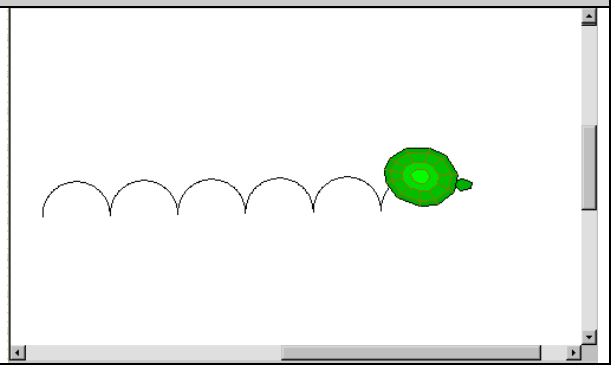
// Festons
 	
VE *// Efface l'écran*
 	
REPETE *500* **[**
  AV *1* **TD** *2*
  SI CAP=*180* **[FCAP** *0***]**
]
	

Figure 11 : Festons.

Instructions	Résultat

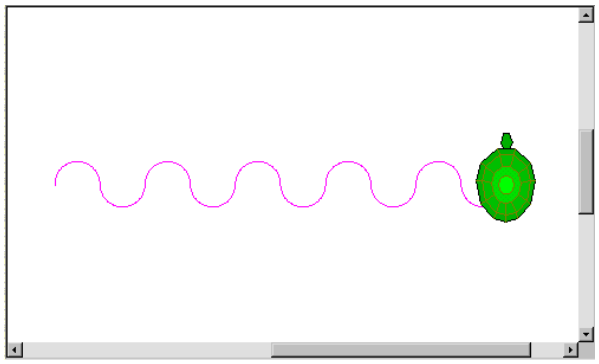
//Serpentins
 	
VE *// Efface l'écran*
FCC *5 // Fixe la couleur du crayon (5 = violet)*
 	
REPETE *10* **[**
  SI CAP=*0* **[**
  REPETE *60* **[AV** *1* **TD** *3***]**
  ]
  [
  REPETE *60* **[AV** *1* **TG** *3* **]**
  ]
]
	

Figure 12 : Serpentins

7.) La boucle « TantQue ».

La boucle « TantQue[5] » permet de répéter un ensemble d'instructions tant qu'une condition est vérifiée.

Ordre	Fonction	Synonymes
TANQUE *exp* **[** *liste* **]**	Répète les instructions contenues dans liste tant que exp est vrai.	

Tableau 6 : La boucle "TantQue".

Instructions	Résultat

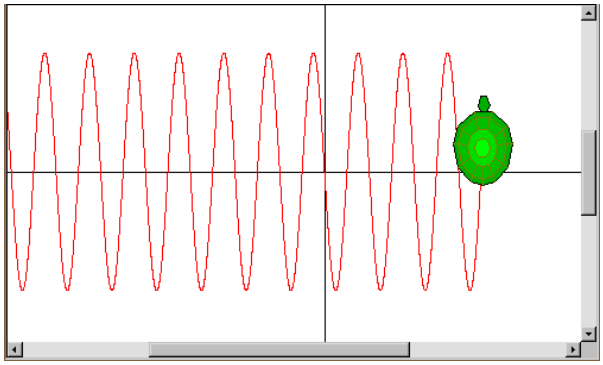
VE *// Efface l'écran*
FCC *0* **//** *Fixe la couleur du crayon (5 = violet)*
DONNE *"x (-80)*
DONNE *"y 0*
LC
FPOS [*-800 0* **] BC FPOS [***800*5 0* **] LC**
FPOS [*0 (-500)* **] BC FPOS [***0 500* **] LC**
FCC *1*
TANTQUE *:x < 80*
[
  DONNE *"y (sin :x) * 100*
  FPOS [*:x*6 :y***]**
  DONNE *"x :x + 0.01* **//** *Pas de 0.01*
  BC
]
	

Figure 13 : Tracé de la courbe f(x) = 100 * SIN(x).

[5] La boucle « TantQue » n'existe pas dans toutes les versions de Logo. Je l'ai incluse ici car son emploi permet de structurer un programme plus facilement qu'en utilisant uniquement la programmation fonctionnelle.

8.) Calculer avec Logo.

8.1.) La primitive "Ecris".

Comme son nom l'indique, la primitive **ECRIS** permet d'écrire des informations à l'écran.

Ordre	Fonction	Synonymes
ECRIS *exp*	Ecris l'expression exp à l'écran.	**EC** *exp*

Tableau 7 : La primitive "Ecris".

8.2.) Notation "infixée".

Logo comprend des expressions telles que "**AV** *(100*4/2+15^2-3) / 3".*

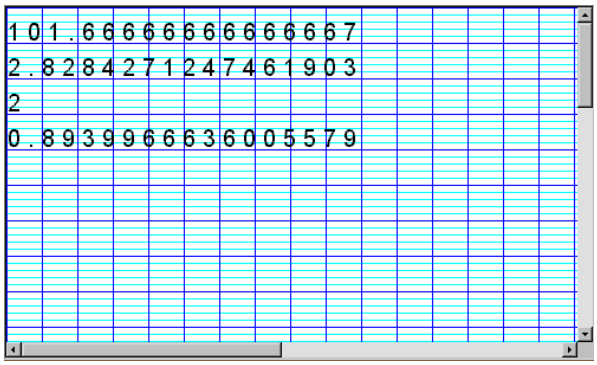
Instructions	Résultat
// Calculs en notation infixe **VE** *// Efface l'écran* **LC** *// Lève le crayon* **FPOS [** *1 28* **]** *// Positionne la tortue* **ECRIS** *(100*3+15*2-5^2) / 3* **FPOS [** *1 58* **]** *// Positionne la tortue* **ECRIS** *2 * SQRT 2* **FPOS [** *1 88* **]** *// Positionne la tortue* **ECRIS** *2* **FPOS [** *1 118* **]** *// Positionne la tortue* **ECRIS** *SIN (45*2)* **CT** *// Cache la tortue*	

Figure 14 : Calculs.

Le tableau suivant donne les symboles mathématiques reconnus par Logo :

Symbole	Fonction
( )	Parenthèses ouvrantes et fermantes.
+ - * /	Addition, soustraction, multiplication et division.
n **!**	factorielle de n.
n **DIV** *m*	Division entière de n par m.
n **MOD** *m*	Reste de la division entière de n par m.
SQRT *n*	Racine carrée de n.
SIN *n*	Sinus de n.
COS *n*	Cosinus de n.
EXP *n*	Exponentielle de n.
LOG *n*	Logarithme de n.
ABS *n*	Valeur absolue de n.
TAN *n*	Tangente de n.
PI	Le nombre PI (3.141516...)

Tableau 8 : Symboles mathématiques.

8.3.) Notation "préfixée".

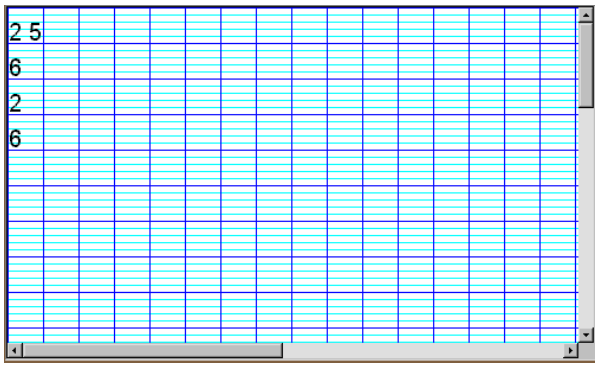
Instructions	Résultat
// Calculs en notation préfixe **VE** *// Efface l'écran* **LC** *// Lève le crayon* **FPOS [** *1 28* **]** *// Positionne la tortue* **ECRIS SOMME** *10 15* **FPOS [** *1 58* **]** *// Positionne la tortue* **ECRIS SOMME SOMME** *1 2 3* **FPOS [** *1 88* **]** *// Positionne la tortue* **ECRIS RESTE** *20 3* **FPOS [** *1 118* **]** *// Positionne la tortue* **ECRIS ENT QUOT** *20 3* **CT** *// Cache la tortue*	

Figure 15 : Calculs.

Attention ! L'expression "**ENT** *5.5 * 2*" renvoie 10 (5*2) tandis que "**ENT** *(5.5 * 2)*" renvoie 11 (la partie entière de "5.5*2").

Ordre	Fonction	Synonymes
SOMME *n m*	Renvoie la somme de n et m.	
DIFF *n m*	Renvoie n - m.	
PROD *n m*	Renvoie le produit de n par m.	**PRODUIT** *n m*
QUOT *n m*	Renvoie n / m.	**QUOTIENT** *n m*
RESTE *n m*	Renvoie le reste de la division entière de n par m	
RC *n*	Renvoie la racine carrée de n.	**RAC** *n*
ENT *n*	Renvoie la partie entière de n.	

8.4.) Les expressions logiques.

Logo peut aussi évaluer une expression logique de ce type :

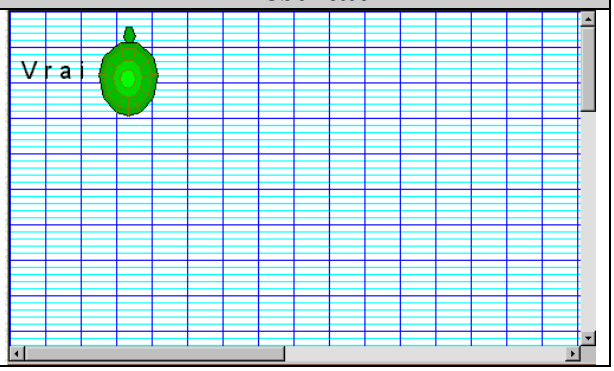
Instructions	Résultat
// *Evaluation d'une expression logique* **VE // *Efface l'écran*** **LC // *Lève le crayon*** **FPOS [** *50 57* **] // *Positionne la tortue*** **ECRIS** *(1/3)*3 = 5-2*2* **FPOS [** *100 57* **] // *Positionne la tortue***	

Figure 16 : Expression logique.

Le tableau suivant donne les opérateurs logiques reconnus par Logo :

Symbole	Fonction
=	Egalité.
>	Strictement supérieur
<	Strictement inférieur
>=	Supérieur ou égal
<=	Inférieur ou égal
&	"ET" logique
|	"OU" logique.

Tableau 9 : Symboles logiques.

9.) Les procédures.

Il faut se rendre à l'évidence : la tortue Logo, cet animal ô combien sympathique, n'est pas ce qu'on appelle une lumière. Bien qu'évoluant dans un environnement hautement technologique (Java, Internet et tout ça), elle possède à peu près autant d'esprit d'initiative qu'un ancien tourne-disque. Alors, il faut tout lui apprendre. Et pour lui apprendre à faire des choses nouvelles on utilise des procédures.

Une procédure, c'est une suite d'instructions à laquelle on a donné un nom. On peut voir une procédure comme une « recette ». Par exemple, la recette pour faire un carré de coté 50 est d'avancer d'une distance de 50, de tourner de 90 degrés à droite, d'avancer d'une distance de 50, de tourner de 90 degrés à droite, d'avancer d'une distance de 50, de tourner de 90 degrés à droite, d'avancer d'une distance de 50, de tourner de 90 degrés à droite. On écrira alors :

```
POUR carre
  REPETE 4 [ AV 50 TD 90 ]
FIN
```

9.1.) Définition et application d'une procédure.

La définition d'une procédure se fait à l'aide des mots clés « **POUR** » et « **FIN**». Par exemple, la procédure suivante

```
POUR cercle
    REPETE 180 [AV 1 TD 2]
FIN
```

signifie « **POUR** *faire un cercle, répète 180 fois les instructions* " **AV** *1* **TD** *2* " , **FIN** *de définition* ». Une fois définie, une procédure s'utilise comme une instruction de base du langage. Voici un exemple d'utilisation de la procédure "*cercle*".

Instructions	Résultat

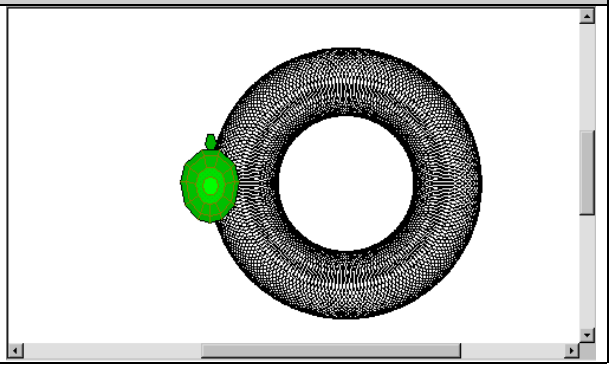
`// Dessin d'un Tore`
 	
`POUR cercle`
`REPETE 180 [AV 1 TD 2]`
`FIN`
 	
`REPETE 180 [cercle AV 4 TD 2]`
	

Figure 17 : Tore.

Dans ce second exemple, on définit une nouvelle procédure appelée "e*toile*", puis on construit une figure géométrique en utilisant cette procédure.

Instructions	Résultat

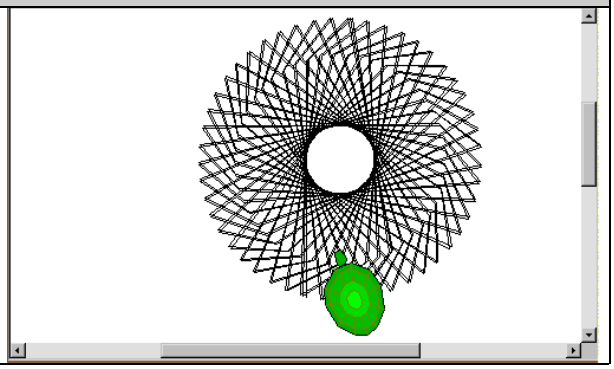
POUR *etoile*
  AV *200*
  TD *45*
  AV *50*
  TD *124*
FIN
 	
REPETE *100* **[** *etoile* **]**
	

Figure 18 : Etoiles.

Voici un troisième exemple de définition et d'utilisation d'une procédure :

Instructions	Résultat

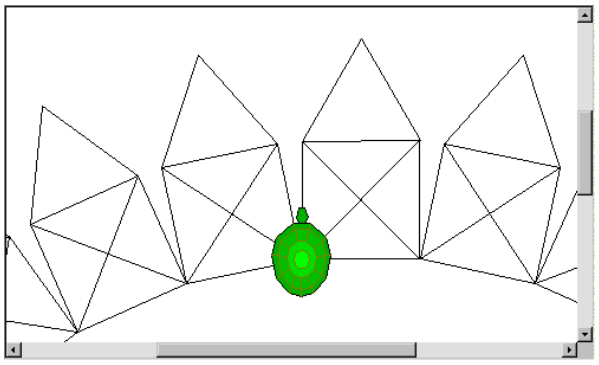
POUR *maison*
  AVANCE *100* **DROITE** *30* **AVANCE** *100*
  DROITE *120* **AVANCE** *100* **DROITE** *120*
  AVANCE *100* **GAUCHE** *135*
  AVANCE *141.42* **GAUCHE** *135*
  AVANCE *100* **GAUCHE** *135*
  AVANCE *141.42* **GAUCHE** *135*
  AVANCE *100* **GAUCHE** *90*
FIN
POUR *maisons*
  REPETE *30* **[** *maison* **TD** *12***]**
FIN
VE *maisons*
	

Figure 19 : Plusieurs maisonnettes.

Et enfin un exemple célèbre tiré du livre de Seymour Papert[6] :

[6] Seymour PAPERT "**Jaillissement de l'esprit** - Ordinateurs et apprentissage" Flammarion 1981.

Instructions	Résultat

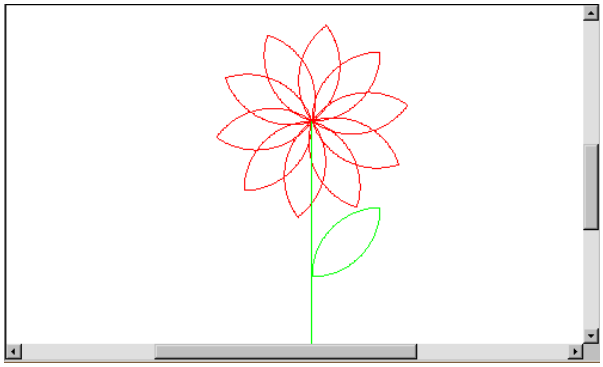
`// Quart de cercle`
`POUR QCERCLE`
`  REPETE 45 [ AV 2 TD 2 ]`
`FIN`
`// Pétale = 2 quarts de cercle`
`POUR PETALE`
`  REPETE 2 [ QCERCLE TD 90 ]`
`FIN`
`// Fleur = 10 pétales`
`POUR FLEUR`
`  REPETE 10 [ PETALE TD 360/10 ]`
`FIN`
`// Plante = fleur + tige + pétale + tige`
`POUR PLANTE`
`  FCC 1 // Rouge`
`  FLEUR`
`  FCC 2 // Vert`
`  RECULE 130 PETALE RECULE 70`
`FIN`
`VE PLANTE CT`
	

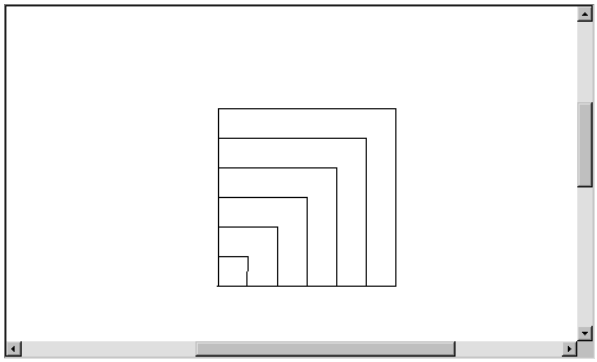
Figure 20 : Procédure "Fleur".

Le programme qui permet de dessiner une fleur fait penser à un jeu de construction. On construit la fleur en assemblant des « briques » élémentaires : un pétale se construit avec deux quarts de cercles, une fleur avec dix pétales, une plante en utilisant une fleur, une tige et un pétale. En cela, ce programme est tout à fait représentatif de la philosophie de Logo : un programme est un assemblage de procédures.

9.2.) Passage de paramètres.

Il est possible de paramétrer les procédures en leur passant des arguments. Dans l'exemple suivant, la procédure "*carre*" attend un nombre en argument. Elle utilisera ce nombre comme longueur d'un coté du carré.

Instructions	Résultat

POUR *carre :cote*
  REPETE *4* **[**
  AV *:cote*
  TD *90*
  ]
FIN
 	
VE
*carre 25*6*
*carre 25*5*
*carre 25*4*
*carre 25*3*
*carre 25*2*
carre 25
CT
	

Ces instructions équivalent à :

REPETE *6* **[***carre 175-25**LOOP* **]**

Figure 21 : Procédure "carre".

Voici un second exemple de procédure paramétrée :

Instructions	Résultat

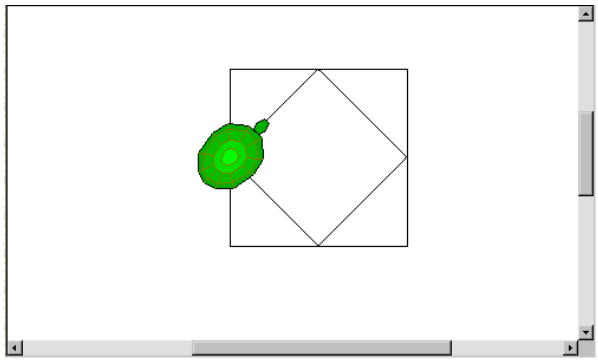
POUR *carre :cote*
  REPETE *4* **[**
  AV *:cote*
  TD *90*
  ]
FIN
POUR *deuxcarres :cote*
  carre :cote
  AV *:cote/2*
  TD *45*
  *carre :cote*0.707*
FIN
VE
deuxcarres 150
	

Figure 22 : Procédure "deuxcarres".

La syntaxe de déclaration d'une procédure peut donc se résumer ainsi :

Ordre	Fonction
POUR *nom_de_procédure :argument_1 :argument_2 ... argument_n* *{ ensemble d'ordres }* **FIN**	La procédure *nom_de_procédure* attend les argument *:argument_1 :argument_2 ... argument_n* et est composée des instructions *{ ensemble d'ordres }.*

Tableau 10 : Définition de procédures.

9.3.) La primitive STOP.

Ordre	Fonction	Synonymes
STOP	Arrêt de la procédure en cours afin de ne pas poursuivre son l'exécution. D'une façon générale, **STOP** rend la main à la procédure appelante.	

Tableau 11 : La primitive "Stop".

Voici 3 exemples d'utilisation de la primitive **STOP** :

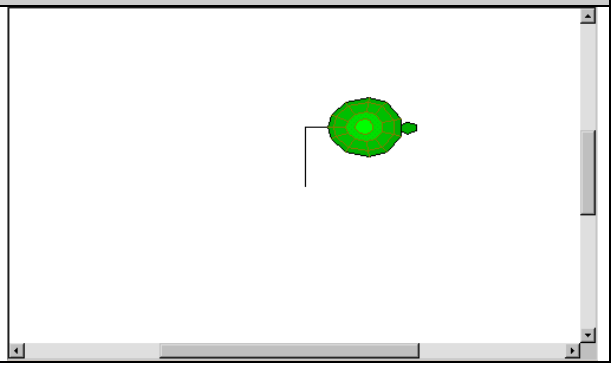
Instructions	Résultat
AV 50 **TD** 90 **AV** 50 **STOP** **FCC** 1 **AV** 50 Le programme s'arrête ici.	

Figure 23 : Utilisation du "STOP" dans un programme.

Dans ce second exemple, l'instruction "**REPETE**" s'arrêtera dés que "**LOOP**" sera supérieur à 100. Par contre, les instructions **FCC** 5 **AV** 50 seront bien exécutée. " **STOP**" provoque la fin d'une boucle.

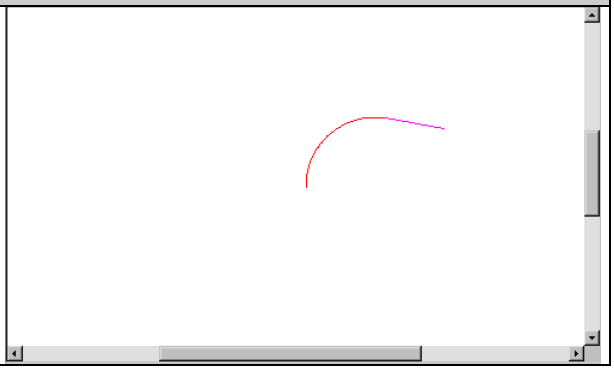
Instructions	Résultat
REPETE 500000 **[** **SI LOOP** > 100 **[STOP] [FCC** 1**]** **AV** 1 **TD** 1 **]** **FCC** 5 **AV** 50 **CT**	

Figure 24 : Sortie d'une boucle avec "STOP".

Dans ce troisième exemple, "**STOP**" provoque l'arrêt de la procédure "figure" :

Instructions	Résultat

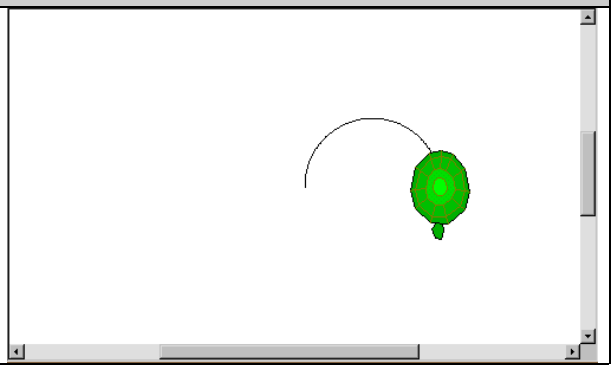
POUR figure
  SI CAP > 180 **[STOP] [AV** 1 **TD** 1**]**
  figure
FIN
VE
figure
	

Figure 25 : Arrêt d'une procédure avec "STOP".

9.4.) Exercices sur les procédures.

Exercice 1.

Ecrire une procédure qui trace à l'écran un polygone de n cotés de longueur l. Evidemment, cette fonction recevra n et l en paramètres.

10.) Les fonctions.

Une fonction est une procédure qui renvoie une valeur.

10.1.) La primitive "Rends".

Ordre	Fonction	Synonymes
RENDS n	Arrêt de la procédure en cours et renvoi de n à la procédure appelante.	**RETOURNE** n **RET** n **RT** n

Tableau 12 : La primitive "RENDS".

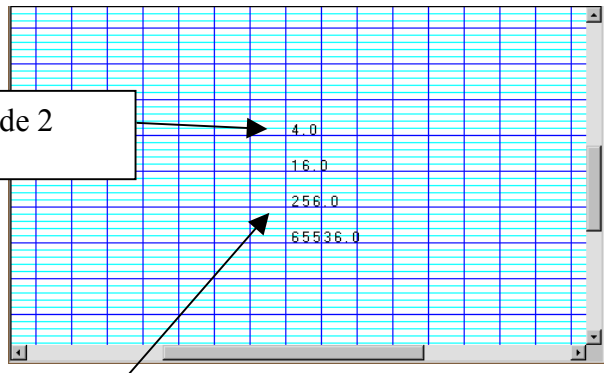
Instructions	Résultat
// Calcul du carré d'un nombre **POUR** carré :c **RET** :c * :c **FIN** **VE** *// Efface l'écran* **FPOS [** -15 10**] ECRIS** carré 2 **FPOS [** -15 40**] ECRIS** carré carré 2 **FPOS [** -15 70**] ECRIS** carré carré carré 2 **FPOS [** -15 100**]** **ECRIS** carré carré carré carré 2 **CT** *// Cache la tortue*	

Figure 26 : Procédure renvoyant une valeur.

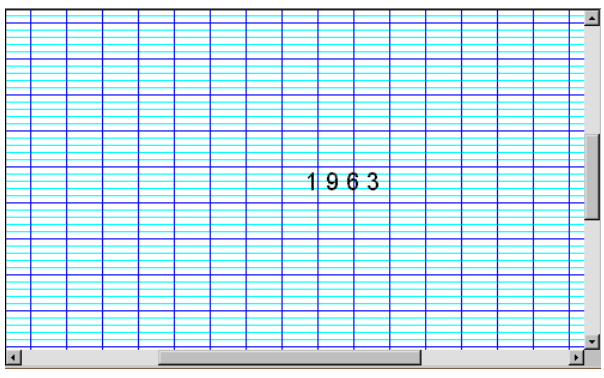
Instructions	Résultat
// Calcul de la surface d'un cercle **POUR** surface :rayon **RET** :rayon * :rayon * **PI** **FIN** **VE** *// Efface l'écran* **ECRIS ENT** surface 25 **CT** *// Cache la tortue*	

Figure 27 : Calcul de surface.

11.) Les Mots.

11.1.) Définition de mots.

Dans le langage Logo, les objets manipulés peuvent être nommés.

Ordre	Fonction	Synonymes
DONNE "nom objet	Donne à l'objet "*objet*" le nom "*nom*".	**RELIE**

Tableau 13 : La primitive "DONNE".

Instruction	Résultat	Observations
ECRIS "Hugo	Hugo	Affiche le mot Hugo
DONNE "Hugo "Victor **ECRIS** :Hugo	Victor	Appelle l'objet "Victor "Hugo Ecris l'objet désigné par Hugo
DONNE "Victor 1802 **DONNE** "Hugo :Victor **ECRIS** :Hugo	1802	Appelle l'objet 1802 Victor Appelle l'objet désigné par Victor Hugo Ecris l'objet désigné par Hugo

Instructions	Résultat

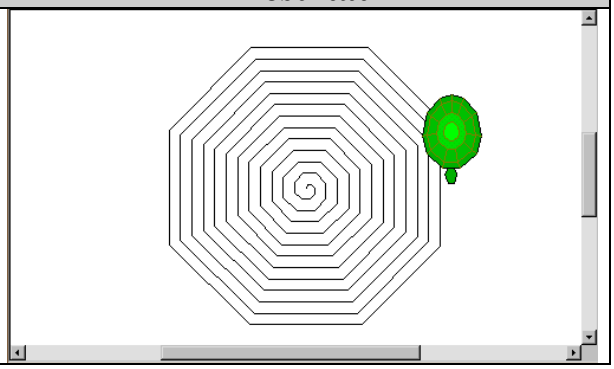
DONNE "valeur 1 // Initialisation
VE
REPETE 150 [
  AV :valeur
  TD 45
  DONNE "valeur :valeur + 1
]
	

Figure 28 : Utilisation simple de "Donne".

11.2.) Noms locaux.

Il est possible de nommer localement (à l'intérieur d'une procédure) des objets avec la primitive LOCALE.

Ordre	Fonction	Synonymes
LOCALE "nom objet	Indique que le nom "nom objet est un nom local à la procédure en cours.	

Tableau 14 : La primitive "LOCALE".

Dans l'exemple suivant, le nom "X est défini globalement et localement. Quand la procédure essai s'exécute, elle affichera la valeur locale de "X. En dehors de cette procédure, c'est la valeur globale de "X qui sera recherchée.

Instructions	Résultat

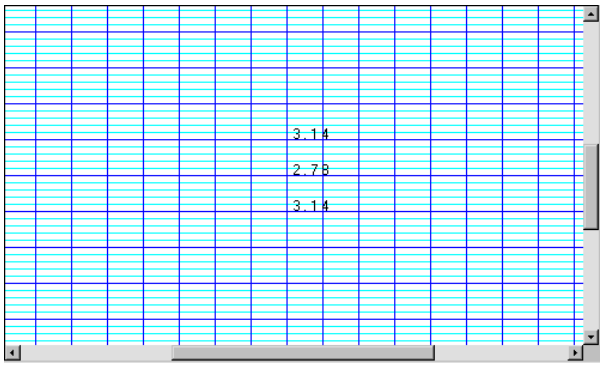
VE CT LC
DONNE "X 3.14
POUR ESSAI
  LOCALE "X
  DONNE "X 2.78
  FPOS [15 40] **ECRIS** :x
FIN
// Ecriture du nom global
FPOS [15 10] **ECRIS** :X
// Ecriture du nom local
ESSAI
// Ecriture du nom global
FPOS [15 70] **ECRIS** :X
	

Figure 29 : Noms locaux.

11.3.) Exercices avec les noms.

Exercice 1.

Pour chacun des programmes ci-dessous, indiquer quel sera le mot écris par la tortue.

ECRIS "Logo	
DONNE "Logo "langage **ECRIS** :Logo	
DONNE "Logo "langage **ECRIS** "Logo	

12.) Les listes.

13.) Récursivité.

13.1.) Procédures récursives.

On appelle procédure récursive toute procédure qui, dans sa définition, fait appel à son propre identificateur. On pourrait, par exemple, définir ainsi la fonction "cercle" :

Instructions	Résultat

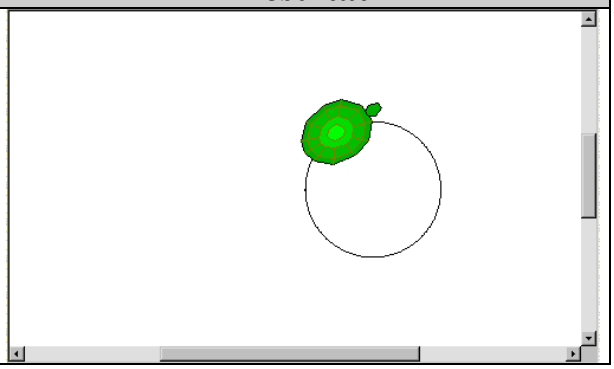
POUR cercle
  AV *1*
  TD *1*
  cercle
FIN
 	
VE
cercle
 	
Ici, la fonction " cercle " s'appelle elle-même.
	

Figure 30 : Définition récursive du cercle.

L'usage de la récursivité permet la création de jolies figures, comme dans l'exemple ci-dessous :

Instructions	Résultat

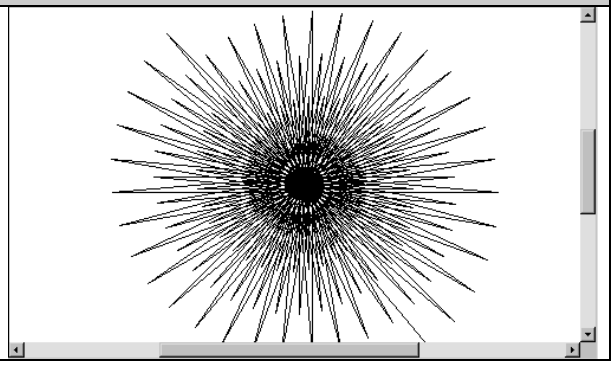
POUR *Spirale1 :cote :angle*
  AV *:cote*
  TD *:angle*
  Spirale1 :cote+2 :angle
FIN
VE
Spirale1 2 185
CT
	

Figure 31 : Procédure "Spirale1".

Instructions	Résultat

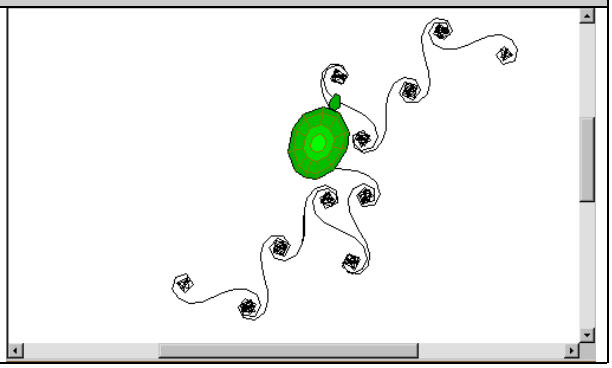
POUR *Spirale2 :dist :angle :inc*
  AV *:dist*
  TD *:angle*
  Spirale2 :dist SOMME :angle **:inc** **:inc**
FIN
VE
Spirale2 10 2 11
	

Figure 32 : Procédure "Spirale2".

En modifiant légèrement les arguments de "Spirale2", on obtient des motifs entièrement différents. Essayez avec :

:dist	:angle	:inc
20	0	10
20	40	30
20	2	11
20	10	80
20	11	80
20	12	80

Instructions	Résultat

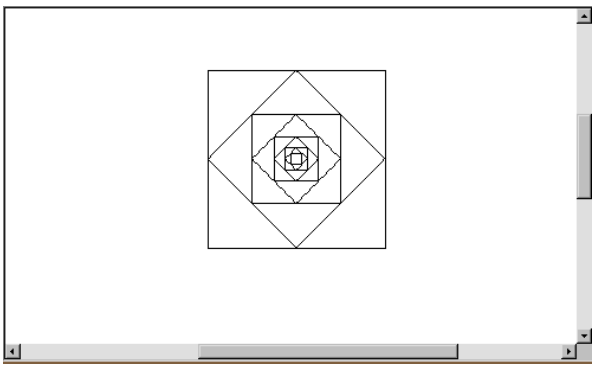
POUR *carre :cote*
  REPETE *4* **[**
  AV *:cote*
  TD *90*
  ]
FIN
POUR *plusieurscarres :cote*
  carre :cote
  SI *:cote<10* **[ ]**
  [AV *:cote / 2*
  TD *45*
  *plusieurscarres :cote*0.707* **]**
FIN
VE
plusieurscarres 150
CT
	

Figure 33 : Procédure "plusieurscarres".

Instructions	Résultat

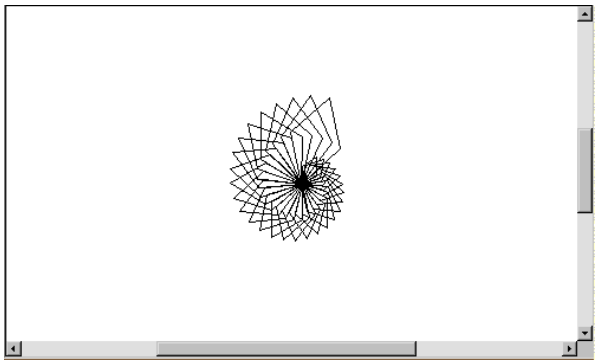
POUR *losange :cote :angle*
  REPETE *2* **[** **AV** *:cote* **TD** *:angle*
  AV *:cote* **TD** *180 - :angle* **]**
FIN
 	
POUR *Spirelo :cote :angle*
  losange *:cote :angle*
  TD *:angle / 5*
  SI CAP=*0* **[STOP]**
  [*Spirelo :cote + 1 :angle* **]**
FIN
VE
spirelo 15 60
CT
	

Figure 34 : Procédure "Spirelo".

13.2.) Fonctions récursives.

De la même façon que les procédures, les fonctions peuvent, elles aussi, être récursives. On pourrait ainsi définir la fonction ***sigma***, qui calcule la somme des entiers entre 0 et n, par :

$$sigma(n) = \begin{cases} 0 \quad si \quad n = 0 \\ \\ n + sigma(n-1) \quad si \quad n > 0 \end{cases}$$

Ce qui s'écrirait, en Logo :

Instructions	Résultat

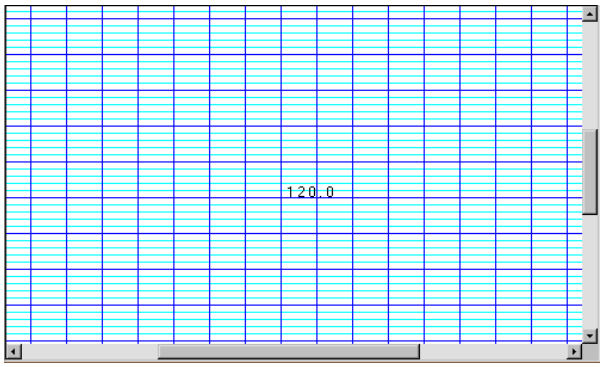
POUR *sigma :n*
  SI *:n = 0*
  [RET *0* **]**
  [RET *:n + sigma :n - 1* **]**
FIN
 	
VE CT
FPOS [*-15 10* **]**
 	
// La somme des nombres entre 0 et 15
// c'est à dire 0+1+2+3+4+5+...+13+14+15
ECRIS *sigma 15*
	

Figure 35 : Fonction "sigma".

En utilisant toujours la récursivité, on pourrait définir la multiplication de x par n (n étant un entier) de la manière suivante :

$$multiplication(x,n) = \begin{cases} 0 \quad si \quad n = 0 \\ \\ x + multiplication(x, n-1) \quad si \quad n > 0 \end{cases}$$

(ce qui revient à calculer $\underbrace{x + x + x + x + x + ... + x}_{n \ fois}$) et cela s'écrirait, en Logo :

Instructions	Résultat

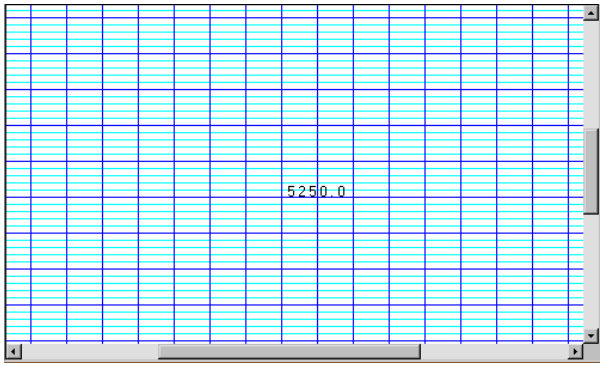
POUR *multiplication :x :n*
  SI *:n = 0*
  [RET *0* **]**
  [RET *:x + multiplication :x (:n - 1)* **]**
FIN
 	
VE CT LC
FPOS [*-15 10* **]**
 	
// Le produit de 150 par 35, c'est à dire
// 150+150+150+150...+150 (35 fois)
ECRIS *multiplication 150 35*
	

Figure 36 : Définition récursive de la multiplication.

Ce dernier exemple est connu de tous les étudiants en informatique. Il s'agit du calcul du PGCD de deux nombres entier a et b en utilisant l'algorithme d'Euclide :

$$p\gcd(a,b) = \begin{cases} a \quad si \quad a = b \\ \\ p\gcd(a-b,b) \quad si \quad a > b \\ \\ p\gcd(a,b-a) \quad si \quad a < b \end{cases}$$

Instructions	Résultat

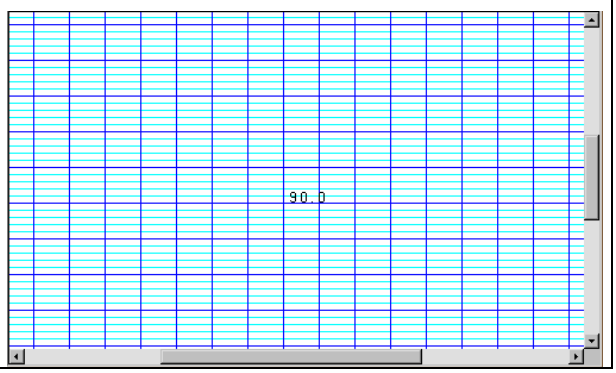
POUR *pgcd :a :b*
  SI *:a = :b*
  [RET *:a* **]**
  [SI :a > :b [RET pgcd :a-:b :b]
[RET pgcd :a :b- :a]
]
FIN
 	
EC *pgcd 1800 2790*
	

Figure 37 : Calcul du PGCD de deux entiers.

13.3.) Exercices sur la récursivité.

Exercice 1.

Calculer le $n_{ième}$ nombre de Lucas défini par :

$$L_1 = 1,\ L_2 = 3 \text{ et } L_n = L_{n-1}+L_{n-2}$$

14.) Fractales.

Instructions	Résultat

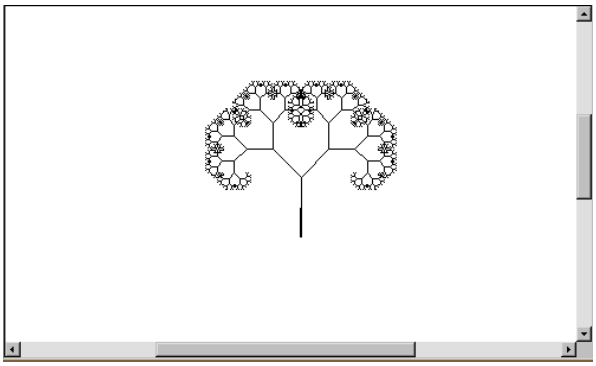
POUR *arbre :L :G*
  SI *:G=0* **[ ]**
  [AV *:L* **TD** *45*
  arbre :L/1.5 :G-1
  TG *90*
  arbre :L/1.5 :G-1
  TD *45*
  REC *:L* **]**
FIN
VE
arbre 50 10
CT
	

Figure 38 : Procédure "arbre".

Instructions	Résultat

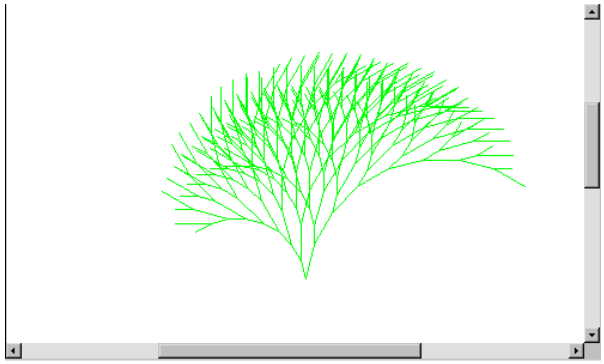
POUR *arbre :l :a :o :f1 :f2*
  SI *:o=0* **[STOP]**
  [GAUCHE *:a* **AVANCE** *:l*
  arbre *:l*:f1 :a :o-1 :f1 :f2*
  LC RECULE *:l* **DROITE** *2*:a* **BC**
  AVANCE *:l * :f2*
  *arbre :l * :f1 :a :o-1 :f1 :f2*
  LC RECULE *:l * :f2* **GAUCHE** *:a* **BC**
  ]
FIN
VE
FCC *2*
arbre 15 15 8 1 2
CT
	

Figure 39 : Arbre penché.

Instructions	Résultat

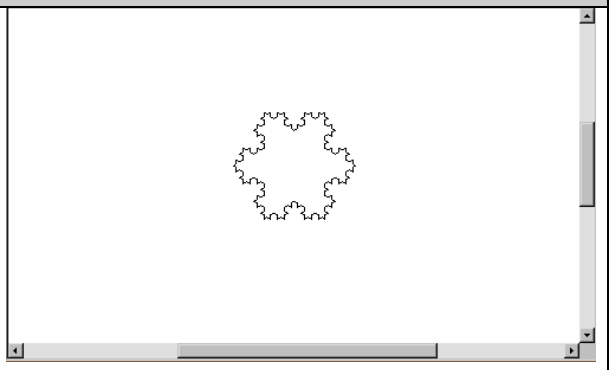
POUR *Khor :L*
  SI *:L < 5* **[AV** *:L* **]**
  [*Khor :L / 3* **TG** *60*
  Khor :L / 3 **TD** *120*
  Khor :L / 3 **TG** *60*
  Khor :L / 3 **]**
FIN
VE
Repete *3* **[** *Khor 90* **TD** *120* **]**
CT
	

Figure 40 : Procédure "Khor".

15.) Encore des procédures...

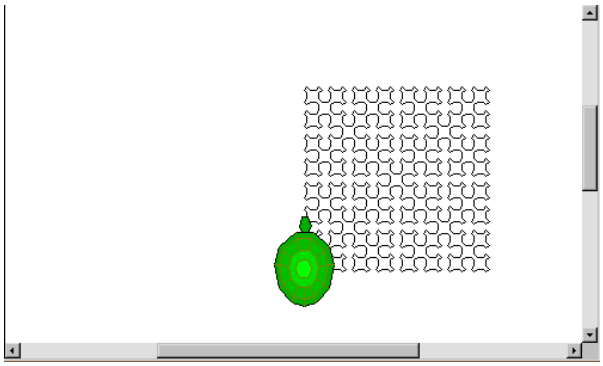
Instructions	Résultat
POUR *corner :SIZE* **DR** *45* **AV** *:SIZE* **DR** *45* **FIN** **POUR** *one_side :SIZE :DIAG :LEVEL* **SI** *:LEVEL = 0* **[stop]** *one_side :SIZE :DIAG :LEVEL - 1* **DR** *45* **AV** *:DIAG* **DR** *45* *one_side :SIZE :DIAG :LEVEL - 1* **GA** *90* **AV** *:SIZE* **GA** *90* *one_side :SIZE :DIAG :LEVEL - 1* **DR** *45* **AV** *:DIAG* **DR** *45* *one_side :SIZE :DIAG :LEVEL - 1* **FIN** **POUR** *sierp :SIZE :LEVEL* **DONNE** *"DIAG :SIZE / sqrt 2* **repete** *4* **[***one_side :SIZE :DIAG :LEVEL* *corner :DIAG***]** **FIN** **VE** *sierp 5 4*	

Figure 41 : Procédure "Sierp".

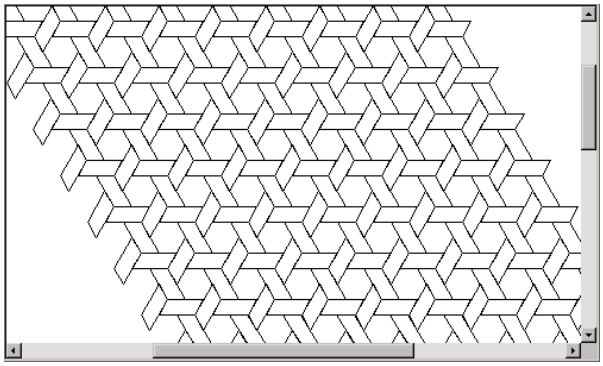
Instructions	Résultat
POUR *skip :SIZE* **LC AV** *:SIZE * 1.5* **BC** **FIN** **POUR** *parallel :SIZE* **REPETE** *2* **[AV** *:SIZE* **DR** *120* **AV** *:SIZE / 2* **DR** *60***]** **FIN** **POUR** *tri :SIZE* **REPETE** *3* **[***parallel :SIZE* **DR** *120***]** **FIN** **POUR** *return :SIZE :LENGTH* **REPETE** *:LENGTH* **[***tri :SIZE* **GA** *60 skip :SIZE* **DR** *60***]** *tri :SIZE* **FIN** **POUR** *along :SIZE :LENGTH* **REPETE** *:LENGTH* **[***tri :SIZE* **DR** *120 skip :SIZE* **GA** *120***]** *tri :SIZE* **FIN** **POUR** *pattern1 :SIZE :LENGTH* *along :SIZE :LENGTH* *skip :SIZE return :SIZE :LENGTH* *skip :SIZE* **FIN** **POUR** *lattice :SIZE :LENGTH :DEPTH* **LC FPOS** *[-100 (-75)* **] GA** *30* **BC** **REPETE** *:DEPTH* **[***pattern1 :SIZE :LENGTH***]** **FIN** *lattice 30 8 4*	

Figure 42 : Procédure "Lattice".

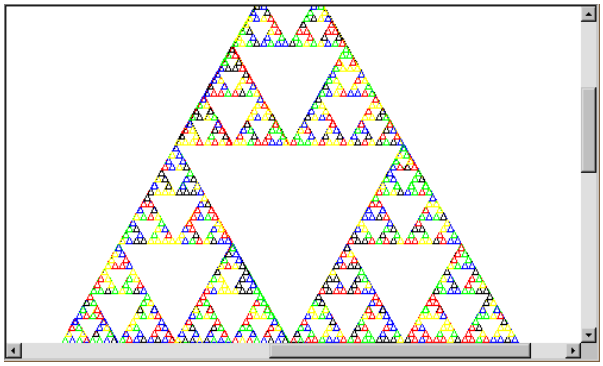
Instructions	Résultat
POUR *triangle :n* **SI** *:n < 4* **[STOP]** **COULEUR HASARD** *5* **REPETE** *3* **[AV** *:n* **DR** *120***]** **DONNE** *"t :t + 1* *triangle :n / 2* **LC AV** *:n / 2* **BC** *triangle :n / 2* **LC DR** *120* **AV** *:n / 2* **GA** *120* **BC** *triangle :n / 2* **LC DR** *240* **AV** *:n / 2* **DR** *120* **BC** **FIN** **DONNE** *"t 0* **VE** **LC FPOS [** *0 0***]** **BC** **DR** *30* *triangle 384* **CT**	

Figure 43 : Procédure "Triangle".

16.) Récapitulatif des primitives.

16.1.) Construction de fonctions.

Nom des primitives	Arguments	Rôle des primitives
CHOSE	NOM	Retourne ce que contient NOM
DONNE, RELIE	NOM OBJ	donne à OBJ un NOM
ECRIS, EC	OBJ	Affiche le contenu de OBJ (et change de ligne)
FIN		Définit la fin d'une fonction utilisateur
LOCALE	NOM	Limite la définition de NOM à la fonction en cours
POUR	NOM	Définie le nom d'une nouvelle fonction et ses entrées.
REPETE	NB LISTE	Exécute NB fois LISTE
RETOURNE, RET, RT	OBJ	Met OBJ à la disposition de la fonction en cours
SI	PR LIST1 LIST2	Exécute LIST1 si PR est vrai, sinon exécute LIST2
STOP		Arrête la fonction en cours.
TANTQUE	PR LIST1	Exécute LIST1 tant que PR est vrai

Tableau 15 : Primitives de construction de fonctions

16.2.) Gestion des objets.

Nom des primitives	Arguments	Rôle des primitives

Tableau 16 : Primitives de gestion des objets

16.3.) Graphismes.

Nom des primitives	Arguments	Rôle des primitives
AV, AVANCE	NB	Commande à la tortue d'avancer de NB pas.
BC, BAISSECRAYON		Actionne le tracé du crayon
CT, CACHETORTUE		Cache la tortue
FCAP, FIXECAP	NB	Oriente la tortue vers NB degrés.
FCC, COULEUR		Fixe la couleur du crayon.
FIXEXY	NB NB	Positionne la tortue au point de coordonnées (NB, NB)
FX, FIXEX	NB	Place la tortue au point d'abscisse NB
FY, FIXEY	NB	Place la tortue au point d'ordonnée NB
FPOS	[NB NB]	Positionne la tortue au point de coordonnées (NB, NB)
LC, LEVECRAYON		Supprime le tracé du crayon.
MT, MONTRETORTUE		Rends la tortue visible.
REC, RECULE, RE	NB	Recule la tortue d'une distance NB
TD, DROITE, DR	NB	Tourne la tortue de NB degrés vers la droite.
TG, GAUCHE, GA	NB	Tourne la tortue de NB degrés vers la gauche.
VE, NETTOIE		Efface l'écran
XCOR		Retourne l'abscisse de la tortue
YCOR		Retourne l'ordonnée de la tortue

Tableau 17 : Primitives graphiques

16.4.) Mathématiques.

Nom des primitives	Arguments	Rôle des primitives

Tableau 18 : Primitives mathématiques

16.5.) Gestion de la mémoire.

Nom des primitives	Arguments	Rôle des primitives

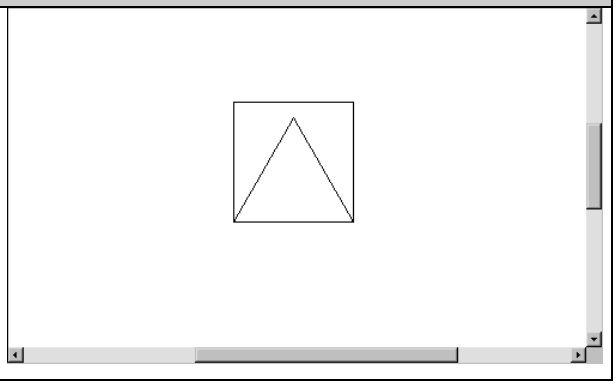
Tableau 19 : Primitives de gestion de la mémoire

17.) Solutions des exercices.

17.1.) Géométrie de la tortue.

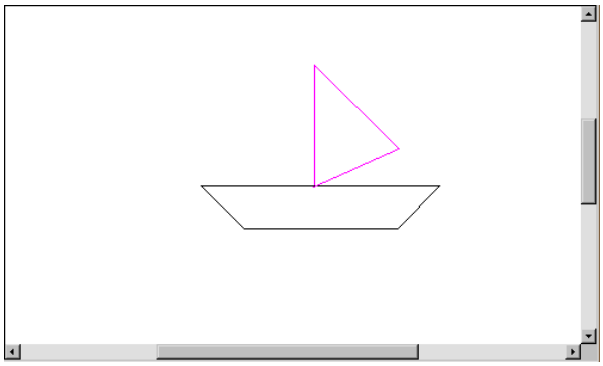
Exercice 1.

Dessiner un carré de coté=100 à l'aide de la tortue. Dessiner ensuite un triangle à l'intérieur de ce carré.

Instructions	Résultat
ve *// Le carré* **av** *100* **td** *90* **av** *100* **td** *90* **av** *100* **td** *90* **av** *100* **td** *90* *// Le triangle* **td** *30* **av** *100* **td** *120* **av** *100* **ct**	

Exercice 2.

Dessiner un bateau à l'aide de la tortue.

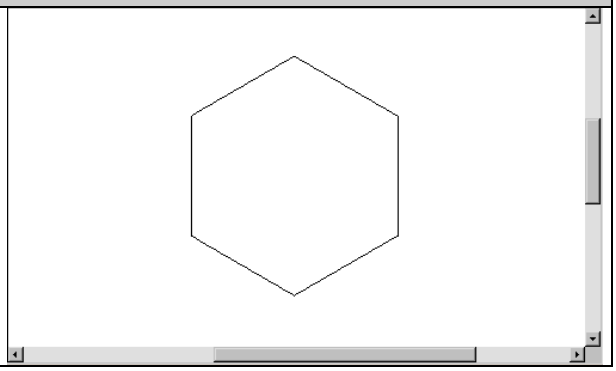
Instructions	Résultat
VE *// Dessin de la coque* **FCC** *0* **TG** *90* **AV** *50* **TD** *45* **AV** *50* **TD** *135* **AV** *200* **TD** *135* **AV** *50* **TD** *45* **AV** *100* *// Dessin de la voile* **FCC** *2* **LC REC** *30* **TD** *90* **AV** *37* **BC** **AV** *100* **TD** *135* **AV** *100* **TD** *111* **AV** *78* **CT**	

17.2.) Exercices sur la répétition.

Exercice 1.

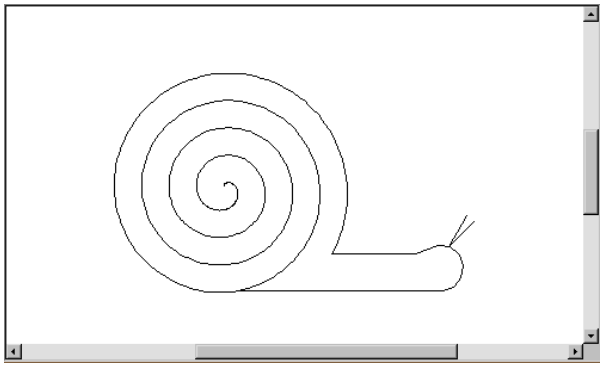
Dessiner un hexagone (une figure géométrique ayant 6 cotés) en utilisant l'ordre **REPETE**.

Instructions	Résultat

VE
REPETE *6*
[
  AV *100*
  TD *360 / 6*
]
CT
	

Exercice 2.

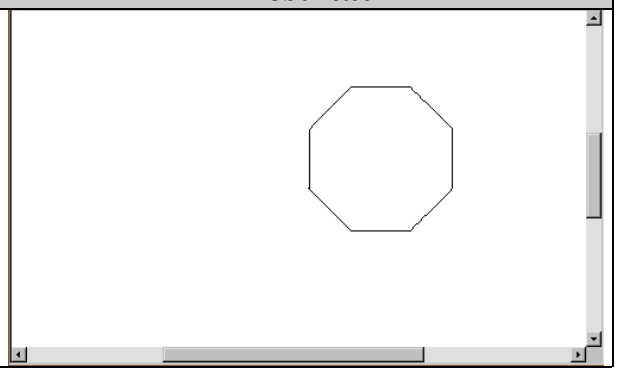
En s'inspirant de la spirale, dessiner un escargot.

Instructions	Résultat
VE // *Efface l'écran* // *Coquille* **REPETE** *550* **[AV** *0.01* *** LOOP TD** *3* **]** // *Corps* **TG** *120* **AV** *70* **TG** *20* **AV** *20* **REPETE** *3* **[TD** *30* **AV** *10* **TD** *30* **AV** *10* **]** **TD** *20* **AV** *170* // *Antennes* **LC REC** *175* **TD** *90* **AV** *37* **BC TD** *30* **AV** *30* **REC** *30* **TD** *15* **AV** *30* **CT**	

17.3.) Exercices sur les procédures.

Exercice 1.

Ecrire une procédure qui trace à l'écran un polygone de n cotés de longueur l. Evidemment, cette fonction recevra n et l en paramètres.

Instructions	Résultat
POUR *polygone :n :l* **REPETE** *:n* **[AV** *:l* **TD** *360/ :n***]** **FIN** **VE** *polygone 8 50* **CT**	

17.4.) Exercices avec les noms.

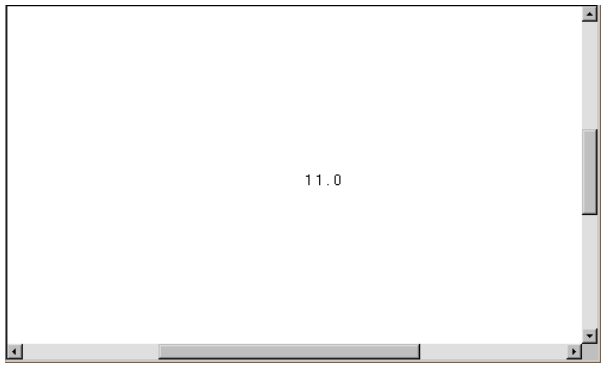
17.5.) Exercices sur la récursivité.

Exercice 1.

Calculer le $n_{ième}$ nombre de Lucas défini par :

$$L_1 = 1,\ L_2 = 3 \text{ et } L_n = L_{n-1}+L_{n-2}$$

Instructions	Résultat

Pour *lucas :n*
SI *:n=1* **[RET** *1***]**
[
  SI *:n=2* **[RET** *3***]**
  [
  RET *(lucas :n-1) + (lucas :n-2)*
  ]
]
FIN
EC *lucas 5*
CT
	

Dans cet exemple, les parenthèses sont importantes. En effet, l'interprèteur pourrait confondre « **RET** *lucas :n-1 + lucas :n-2* » avec « **RET** *lucas ((:n-1) + lucas :n-2)* » ou « **RET** *lucas :n- (1 + lucas :(n-2))* », ce qui n'est évidemment pas la même chose !

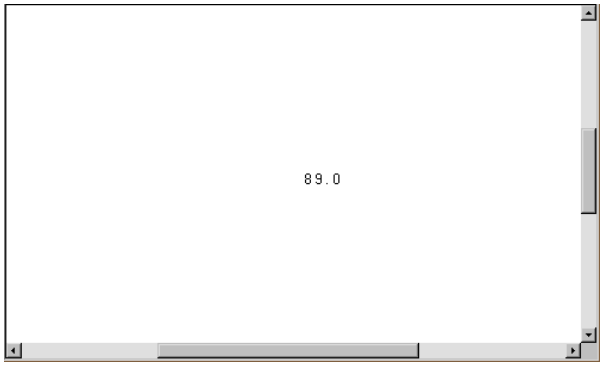
Exercice 2.

Léonard de Pise, également appelé Léonardo Fibonacci, présenta en 1202 le problème suivant : "*Un homme a placé un couple de lapins dans un enclos. Combien de couples de lapins peuvent être produits par celui-ci en un an si on suppose que tous les mois chaque couple engendre un nouveau couple lequel devient alors productif à partir du second mois ?*". Le nombre de lapins progresse selon une suite récurrente dite "de Fibonacci" définie par :

$$U_1 = 1,\ U_2 = 2 \text{ et } U_n = U_{n-1}+U_{n-2}$$

Calculez le nième terme de la suite de Fibonacci.

Instructions	Résultat

Pour *Fibonacci :n*
SI *:n=1* **[RET** *1***]**
[
  SI *:n=2* **[RET** *2***]**
  [
  RET *(Fibonacci :n-1) + (Fibonacci :n-2)*
  ]
]
FIN
EC *Fibonacci 10*
CT
	

18.) Bibliographie.

Seymour PAPERT "**Jaillissement de l'esprit** - Ordinateurs et apprentissage" Flammarion 1981.

Boris ALLAN "**Le Logo**" BELIN 1984.

André MYX et Pierre SUBTIL « **Logo, votre langage de programmation** » CEDIC NATHAN 1985.

X. LEROY « **Logo, langage pour tous** » MICRO SYSTEMES 1985.

19.) Tables.

19.1.) Tables des matières.

19.2.) Tableaux.

19.3.) Figures.

20.) Index des mots clé.